Object-Oriented Programming I (CSE 121)
Week # 07
(Introduction to OOP in C++)

Objective(s)

In this lecture, you will learn:

1. What is OOP?
2. C++ class and object
3. How to declare class & objects
4. How to access class members
5. How to add function to a class
6. How to call the function
7. How to add parameterized function into a class
8. this keyword
9. Constructor and types of constructor
10. Destructor
11. Access specifiers or modifiers
12. Selection operators

Introduction to OOP in C++

(class, object, attributes, methods, this, constructor, destructor, access specifiers, selection operators)

What is OOP?

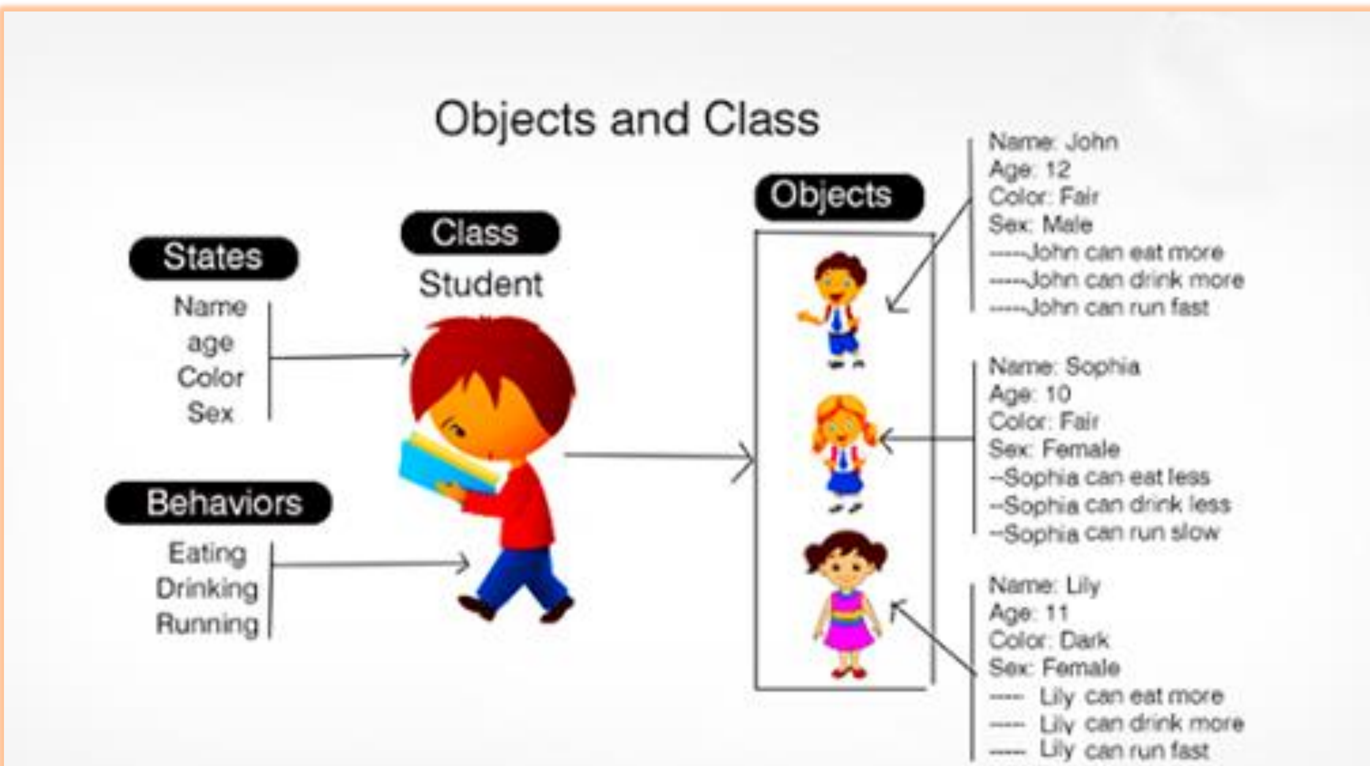
OOP stands for Object-Oriented Programming. **OOP** is a methodology or paradigm to design a program using **classes** and **objects**. In other words, object-oriented programming is about creating **objects** that contain both data and functions.

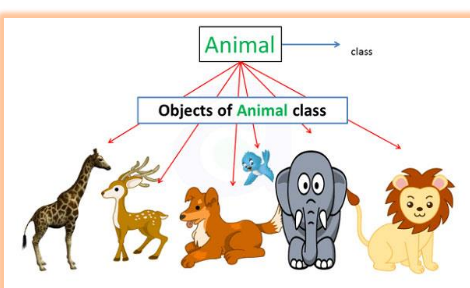
Apart from **objects** and **classes**, OOP has four core concepts which are listed below:

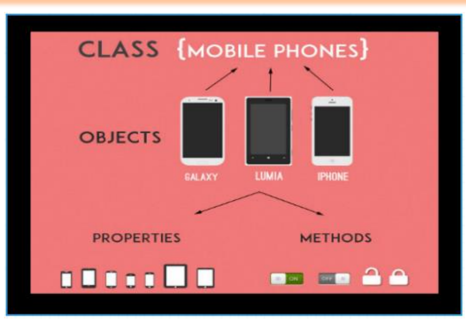
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

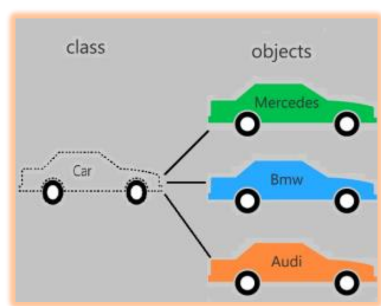
C++ Classes and Objects

Classes and **objects** are the two main aspects of object-oriented programming. Look at the following illustrations to see the difference between class and objects:









Object

An **object** is a real-world entity such as mobile, car, chair, table, pen etc. that has states/attributes and behaviors/methods. Here, states mean data/variables and behaviors mean functions. These are often referred to as "class members".

Class

A **class** is a template from which objects are created. In other words, **class** is a user-defined data type that we can use in our program, and it works as a "blueprint" for creating **objects**. So, we can say that class is the collection of the similar kind of objects

Differentiate between class and object

No.	Class	Object
1.	Class is a mechanism/template of binding data members and associated methods in a single unit.	Object is an instance/blueprint of a class. All the members of the class can be accessed through object.
2.	It is logical existence.	It is physical existence.
3.	Memory space is not allocated, when it is created.	Memory space is allocated, when it is created.
4.	Definition is created once.	It is created many times as you require.

Create a Class

To create a class, use the class keyword:
Create a class called "Student":

```
class Student{     // The class
  public:          // Access specifier
    int sID;       // Attribute (int variable)
    string sName; // Attribute (string variable)
};
```

Example explained

- The class keyword is used to create a class called Student.
- The public keyword is an **access specifier**, which specifies that members (attributes and methods) of the class are accessible from outside the class.
- Inside the class, there is an integer variable sID and a string variable sName. When variables are declared within a class, they are called **attributes**.
- At last, end the class definition with a semicolon ;.

Create an Object

- In C++, an object is created from a class. We have already created the class named Student, so now we can use this to create objects.
- To create an object of Student, specify the class name, followed by the object name.
- To access the class attributes (sId and sname), use the dot syntax (.) on the object.

Example 1: class and object

Create an object called "obj" and access the attributes:

```cpp
#include <iostream>
using namespace std;

class Student {     // The class
  public:           // Access specifier
    int sID;        // Attribute (int variable)
    string sName;  // Attribute (string variable)
};

int main() {
  Student obj;  // Create an object of Student

  // Access attributes and set values
  obj.sID = 1001;
  obj.sName = "Bulbul Ahamed";

  // Print attribute values
  cout << obj.sID << "\n";
  cout << obj.sName << "\n";
  return 0;
}
```

```
C:\Users\Bulbul\Desktop\OOP\ex1.exe
1001
Bulbul Ahamed

Process returned 0 (0x0)   execution time : 0.093 s
Press any key to continue.
```

Multiple Objects

Example 2: You can create multiple objects of one class.

```cpp
#include <iostream>
using namespace std;

class Student {     // The class
  public:           // Access specifier
    int sID;        // Attribute (int variable)
    string sName;  // Attribute (string variable)
};

int main() {
  Student obj1;  // Create an object of Student

  // Access attributes and set values
  obj1.sID = 1001;
  obj1.sName = "Bulbul Ahamed";

  // Print attribute values
  cout << obj1.sID << "\n";
  cout << obj1.sName << "\n";

  Student obj2;  // Create another object of Student

  // Access attributes and set values
  obj2.sID = 1002;
  obj2.sName = "Aleeza Ahamed";

  // Print attribute values
  cout << obj2.sID << "\n";
  cout << obj2.sName << "\n";
  return 0;
}
```

```
C:\Users\Bulbul\Desktop\OOP\ex2.exe
1001
Bulbul Ahamed
1002
Aleeza Ahamed

Process returned 0 (0x0)   execution time : 0.187 s
Press any key to continue.
```

Class Methods

Methods are **functions** that belongs to the class.
There are two ways to define functions that belongs to a class:

- Inside class definition
- Outside class definition

In the following example, we define a function inside the class, and we name it "display".

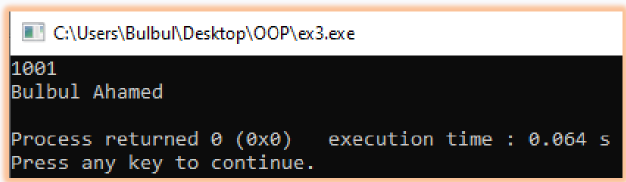
Note: You access methods just like you access attributes; by creating an object of the class and by using the dot syntax (.):

Example 3: Method defined inside the class

```
#include <iostream>
using namespace std;

class Student {        // The class
  public:              // Access specifier
    void display() {   // Method/function defined inside the class
      cout << "1001\n";
      cout << "Bulbul Ahamed\n";
    }
};

int main()
{
  Student obj; //Create an object of Student
  obj.display(); //Call the method
  return 0;
}
```



To define a function outside the class definition, you have to declare it inside the class and then define it outside of the class. This is done by specifying the name of the class, followed the scope resolution :: operator, followed by the name of the function:

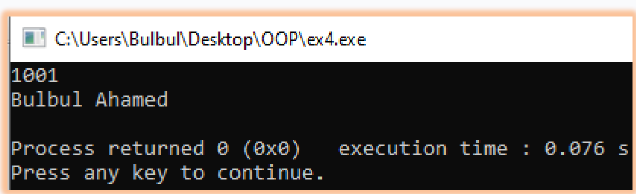
Example 4: Method defined outside the class

```
#include <iostream>
using namespace std;

class Student {        // The class
  public:              // Access specifier
    void display();    // Method/function declaration
};

// Method/function definition outside the class
void Student::display() {
      cout << "1001\n";
      cout << "Bulbul Ahamed\n";
}

int main()
{
  Student obj; //Create an object of Student
  obj.display(); //Call the method
  return 0;
}
```

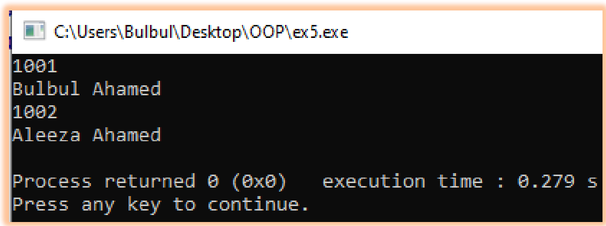


Class Attributes and Methods

Example 5: A class with attributes and method (method defines inside the class)

```
#include <iostream>
using namespace std;
class Student {     // The class
  public:           // Access specifier
    int sID;        // Attribute (int variable)
    string sName;   // Attribute (string variable)
    void display() // Method/function defined inside the class
    {
        cout << sID << "\n" << sName << "\n";
    }
};
int main()
{
    Student obj1; //Create an object of Student
    //Access attributes and set values
    obj1.sID = 1001;
    obj1.sName = "Bulbul Ahamed";
    obj1.display(); //Call the method

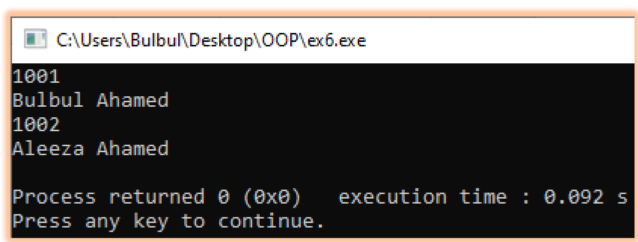
    Student obj2; //Create another object of Student
    //Access attributes and set values
    obj2.sID = 1002;
    obj2.sName = "Aleeza Ahamed";
    obj2.display(); // Call the method
    return 0;
}
```



Example 6: A class with attributes and method (method defined outside the class)

```
#include <iostream>
using namespace std;
class Student {     // The class
  public:           // Access specifier
    int sID;        // Attribute (int variable)
    string sName;   // Attribute (string variable)
    void display(); // Method/function declaration
};
//Method/function definition outside the class
void Student::display() {
  cout << sID << "\n" << sName << "\n";
}
int main()
{
    Student obj1; // Create an object of Student
    //Access attributes and set values
    obj1.sID = 1001;
    obj1.sName = "Bulbul Ahamed";
    obj1.display(); //Call the method

    Student obj2; //Create another object of Student
    //Access attributes and set values
    obj2.sID = 1002;
    obj2.sName = "Aleeza Ahamed";
    obj2.display(); //Call the method
    return 0;
}
```

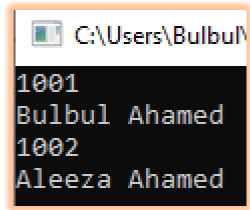


You can also add parameters into the functions.

Example 7: Parameterized function into a class

```
#include <iostream>
using namespace std;

class Student {     // The class
  public:           // Access specifier
    int sID;        // Attribute (int variable)
    string sName;   // Attribute (string variable)
    // Parameterized Method/function defined inside the class
    void setValue(int x, string y){
        sID = x;
        sName = y;
    }
        // Method/function defined inside the class
    void getValue(){
        cout << sID << "\n" << sName << "\n";
    }
};
int main(){
    Student obj1, obj2; //Create two objects of Student
    obj1.setValue(1001,"Bulbul Ahamed"); //Call the method with arguments
    obj1.getValue(); //Call the method
    obj2.setValue(1002,"Aleeza Ahamed"); //Call the method with arguments
    obj2.getValue(); //Call the method
    return 0;
}
```



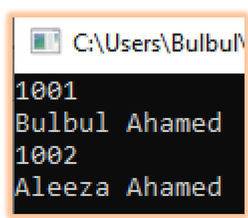
C++ this Keyword

In C++ programming, **this** is a keyword that refers to the current instance of the class.

Example 8: this keyword refers to the attributes of current class

```
#include <iostream>
using namespace std;

class Student {     // The class
  public:           // Access specifier
    int sID;        // Attribute (int variable)
    string sName;   // Attribute (string variable)
    //Parameterized Method/function defined inside the class
    void setValue(int sID, string sName){
        this -> sID = sID;
        this -> sName = sName;
    }
    // Method/function defined inside the class
    void getValue(){
        cout << sID << "\n" << sName << "\n";
    }
};
int main(){
    Student obj1, obj2; //Create two objects of Student
    obj1.setValue(1001,"Bulbul Ahamed"); //Call the method with arguments
    obj1.getValue(); //Call the method
    obj2.setValue(1002,"Aleeza Ahamed"); //Call the method with arguments
    obj2.getValue(); //Call the method
    return 0;
}
```



C++ Constructor

In C++, constructor is a special method/function which is invoked/called automatically when an object of a class is created. To create a constructor, use the same name as the class, followed by parentheses (). There can be three types of constructors in C++:

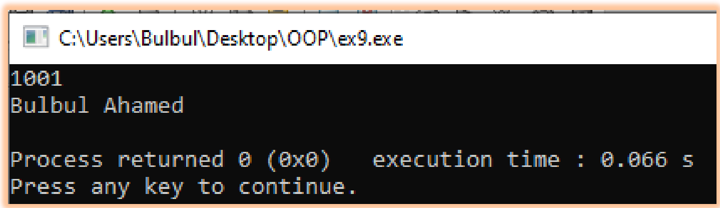
- Default constructor
- Parameterized constructor
- Copy constructor

Default Constructor

A constructor which has no argument/parameter is known as default constructor. It is invoked at the time of creating object.

Example 9: Let's see the simple example of C++ default Constructor.

```
#include <iostream>
using namespace std;
class Student {     // The class
  public:           // Access specifier
    Student() {     // Default constructor
      cout << "1001\n";
      cout << "Bulbul Ahamed\n";
    }
};
int main() {
  Student obj;  // Create an object of Student
  return 0;
}
```

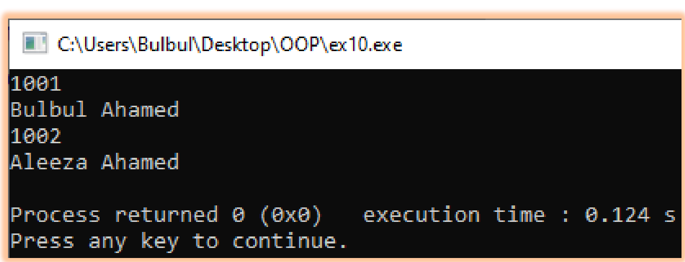


Parameterized Constructor

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

Example 10: A simple example of C++ Parameterized Constructor.

```
#include <iostream>
using namespace std;
class Student {     // The class
  public:           // Access specifier
    int sID;        // Attribute (int variable)
    string sName;   // Attribute (string variable)
    Student(int x, string y){ //Parameterized constructor
        sID = x;
        sName = y;
    }
    void getValue(){
        cout << sID << "\n" << sName << "\n";
    }
};
int main(){
    Student obj1(1001,"Bulbul Ahamed");
    obj1.getValue(); //Call the method
    Student obj2(1002,"Aleeza Ahamed");
    obj2.getValue(); //Call the method
    return 0;
}
```



Note: The constructor has the same name as the class, it is always `public`, and it does not have any return value.

Copy Constructor

A copy constructor is a member function which initializes an object using another object of the same class. In other words, copy constructor is used to declare and initialize an object from another object.

Example 11: A simple example of Copy Constructor

```
#include <iostream>
using namespace std;

class Student {     // The class
private:            // Access specifier
    int sID;        // Attribute (int variable)
    string sName;   // Attribute (string variable)
public:             // Access specifier
        //Parameterized constructor
    Student(int x, string y)
    {
        sID = x;
        sName = y;
    }
    //Copy constructor
    Student(Student &obj)
    {
        cout << "Copy constructor runs successfully.\n";
        sID = obj.sID;
        sName = obj.sName;
    }
    void getValue()
    {
        cout << sID << "\n" << sName << "\n\n";
    }
};
int main()
{
    Student obj1(1001,"Bulbul Ahamed"); //Parameterized constructor is called
    obj1.getValue(); //Call the method

    Student obj2(obj1); //Copy constructor is called
    obj2.getValue(); //Call the method
    return 0;
}
```

```
C:\Users\Bulbul\Desktop\OOPs\CopyCon.exe
1001
Bulbul Ahamed

Copy constructor runs successfully.
1001
Bulbul Ahamed


Process returned 0 (0x0)   execution time : 0.082 s
Press any key to continue.
```

C++ Destructor

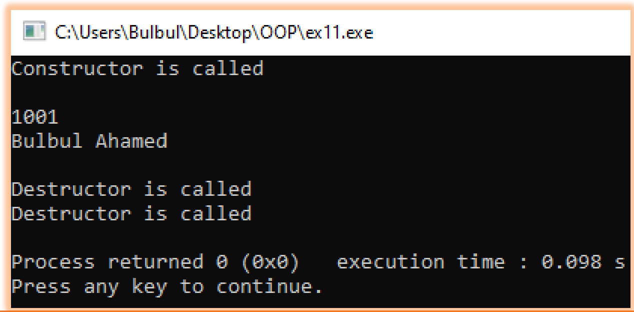
A destructor is defined like constructor and it is invoked/called automatically. It must have same name as class. But it is prefixed with a **tilde** sign (~).

Note: C++ destructor cannot have parameters.

Example 11: A simple example of Constructor and Destructor.

```
#include <iostream>
using namespace std;

class Student {     // The class
  public:           // Access specifier
    int sID;        // Attribute (int variable)
    string sName;  // Attribute (string variable)
    Student(){      //Default constructor
        cout << "Constructor is called\n";
    }
    Student(int x, string y){ //Parameterized constructor
        sID = x;
        sName = y;
    }
    ~Student(){ //Destructor
        cout << "Destructor is called\n";
    }
    void getValue(){
        cout << sID << "\n" << sName << "\n";
    }
};
int main(){
    Student obj;
    Student obj1(1001,"Bulbul Ahamed");
    obj1.getValue(); //Call the method
    return 0;
}
```



Access Specifiers or Modifiers

Access specifiers define how the members (attributes and methods) of a class can be accessed.

In C++, there are three access specifiers:

- public - members are accessible from outside the class
- private - members cannot be accessed (or viewed) from outside the class
- protected - members cannot be accessed from outside the class, however, they can be accessed in inherited/derived classes.

Specifiers	Within same class	In derived class	Outside the class
public	Yes	Yes	Yes
private	Yes	No	No
protected	Yes	Yes	No

Note: If we do not specify any access modifiers for the members inside the class then by default the access modifier for the members will be **Private**.

Selection Operators

In C++, there are two types of selection operators: **dot (.) operator** and **arrow (->) operator.**

- The **dot (.) operator** is called "direct member selection" operator. It is used to access the class members directly via object name.
- The **arrow (->) operator** is called as "indirect member selection" operator. It is used to access the members indirectly with the help of **pointers**.

Example 12: A simple example of selection operators.

```cpp
#include <iostream>
using namespace std;

class Student {     // The class
  public:           // Access specifier
    int sID;        // Attribute (int variable)
    string sName;   // Attribute (string variable)
    void display() {  // Method/function defined inside the class
        cout << sID << endl;
        cout << sName << endl;
    }
};

int main()
{
    Student obj; // Create an object of Student
    // Access attributes and set values
    obj.sID = 1001;
    obj.sName = "Bulbul Ahamed";
    obj.display();//Call the method

    Student obj1; // Create another object of Student
    Student *p = &obj1; // Create a pointer
    // Access attributes and set values
    p -> sID = 1002;
    p -> sName = "Aleeza Ahamed";
    p -> display();//Call the method

    return 0;
}
```

```
C:\Users\Bulbul\Desktop\OOP\ex12.exe
1001
Bulbul Ahamed
1002
Aleeza Ahamed

Process returned 0 (0x0)   execution time : 0.097 s
Press any key to continue.
```

Constructor Overloading in C++

- Making multiple constructors in the same class.
- Overloaded constructors must have the same name with different parameters.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

………………………………